Министерство образования Республики Беларусь

Учреждения образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ

И РАДИОЭЛЕКТРОНИКИ

КАФЕДРА ИНФОРМАТИКИ

Отчет по лабораторной работе №5

По теме «Интерпретация исходного кода»

Выполнил:

студент гр. 053502

Песоцкий В.А.

Проверил:

Гриценко Н. Ю.

Минск 2023

**СОДЕРЖАНИЕ**

[1 Цель работы 3](#_Toc134835291)

[2 Результаты выполнения 4](#_Toc134835292)

[Приложение А. Текст программ 7](#_Toc134835293)

# 1 ЦЕЛЬ РАБОТЫ

На основе результатов анализа лабораторных работ 1-4 выполнить интерпретацию программы.

# 

# 

# 

# 2 РЕЗУЛЬТАТЫ ВЫПОЛНЕНИЯ

Демонстрируется интерпретация кода программ, на различный языковой синтаксис и обработку.

На рисунке 2.1 представлен код первой программы**:**

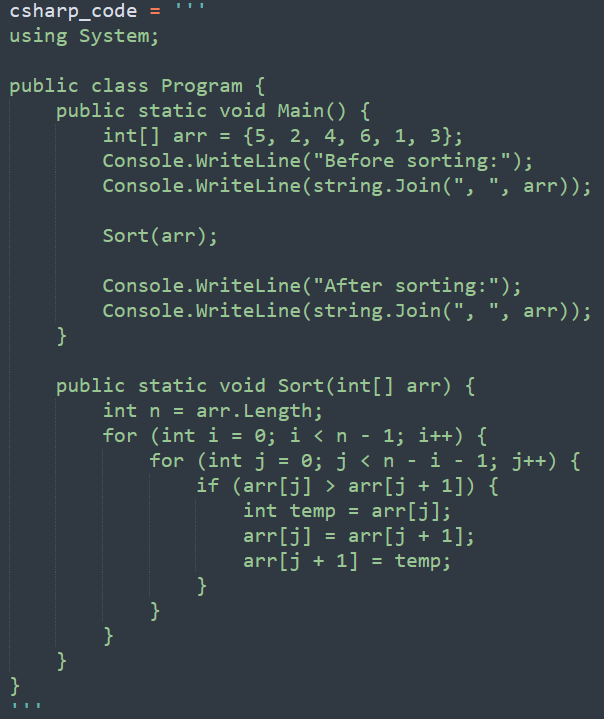


Рисунок 2.1 – Код сортировки пузырьком

Результат интерпретации программы представлен ниже на рисунке 2.2. Программа ожидает когда пользователь введет длину массива, а затем соответсвующее количество значений, после чего сортирует его пузырьком и выводит результат на экран.

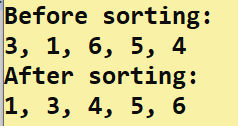


Рисунок 2.2 – Результат выполнения

На рисунке 2.3 представлен код программы, проверяющий работу с байт-элементами:

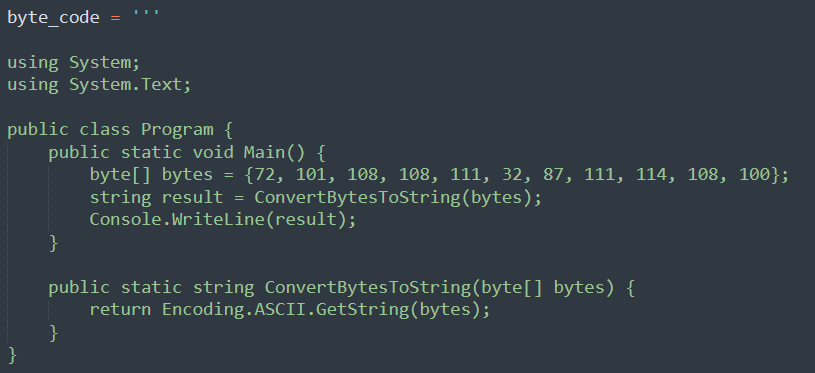


Рисунок 2.3 – Байт-конвертер

Результат интерпретации второй программы представлен ниже на рисунке 2.4. Программа принимает на вход массив из байт, после чего согласно кодировке ASCII, преобразует массив байт в набор из букв латинского алфавита.

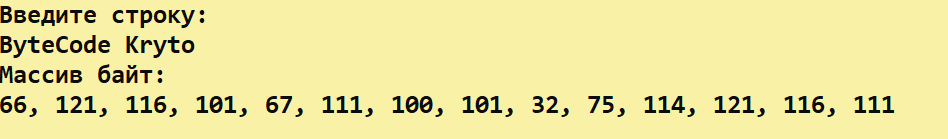


Рисунок 2.4 – Результат выполнения

Далее рассмотрим код программы, проверяющий работу конкатенации строк и работу условных операторов. Данная программа принимает массив элементов, элемент который требуется найти и используя алгоритм бинарного поиска проводит поиск элемента, если элемент найден, то выводит его и его индекс в массиве, если же не найден, то выводит результат отсутствия элемента, чем проверяется условные выражения. Внутри методов библиотеки System, выводящих строки, используется присоединение двух строк, чем проверяется работа методов внутри строк, различение операторов от самой строки. Код представлен на рисунке 2.5:

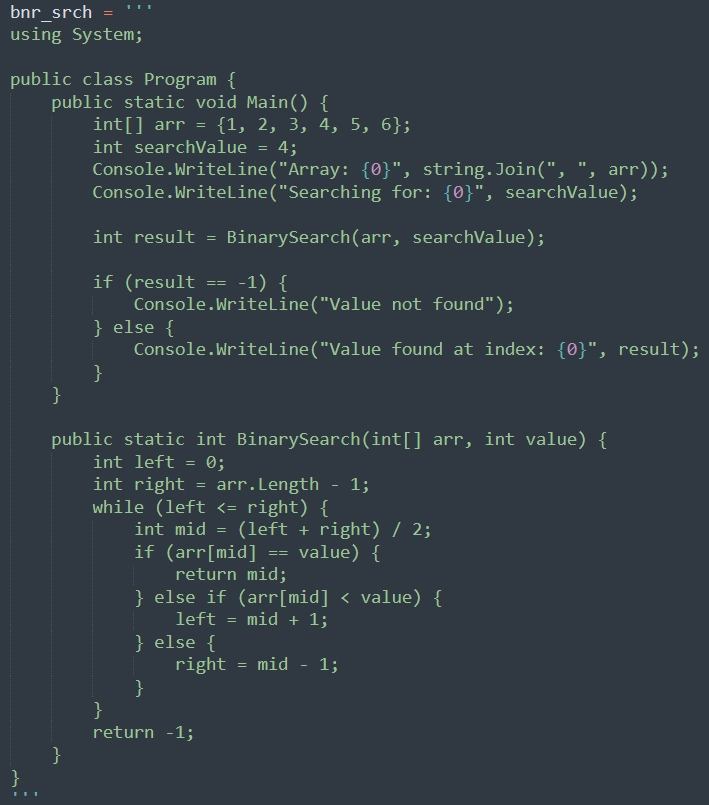


Рисунок 2.5 – Бинарный поиск

Результат интерпретации заключительной программы представлен ниже на рисунке 2.6.

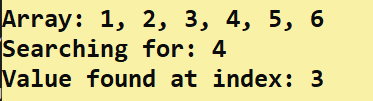
****

Рисунок 2.6 – Бинарный поиск

**ПРИЛОЖЕНИЕ А. ТЕКСТ ПРОГРАММ**

import clr

import System

import os

import lexical

import synttest

pathDLLSort = os.getcwd() + "\\pythonCompile.dll"

pathDLLByte = os.getcwd() + "\\Bytes.dll"

clr.AddReference(pathDLLSort)

clr.AddReference(pathDLLByte)

# Bubble Sort Started

from pythonCompile import BubbleSort

import System

#print(BubbleSort)

print('\n')

print('\n')

print('\n')

arr = [5, 2, 9, 1, 5, 6]

cs\_arr = System.Array.CreateInstance(System.Int32, len(arr))

for i, x in enumerate(arr):

cs\_arr[i] = x

BubbleSort.Sort(cs\_arr)

for i in range(len(arr)):

arr[i] = cs\_arr[i]

#print(arr)

#print(clr.GetClrType(System.Object).Assembly.ImageRuntimeVersion)

# Bubble Sort Ended

#Byte Code Strted

import Bytes

#print(Bytes)

from Bytes import ByteCode

bytes\_c = [104, 101, 108, 108, 111]

string = ByteCode.ConvertBytesToString(bytes\_c)

#print(string) # "hello"

bytes\_b = [77, 121, 32, 78, 97, 109, 101, 32, 105, 115, 32, 86, 108, 97, 100]

string\_2 = ByteCode.ConvertBytesToString(bytes\_b)

#print(string\_2) #My Name is Vlad ...

# пример кода на C#

csharp\_code = '''

using System;

public class Program {

public static void Main() {

Console.WriteLine("Enter array length:");

int n = Convert.ToInt32(Console.ReadLine());

int[] arr = new int[n];

Console.WriteLine("Enter array elements:");

for (int i = 0; i < n; i++) {

arr[i] = Convert.ToInt32(Console.ReadLine());

}

Console.WriteLine("Before sorting:");

Console.WriteLine(string.Join(", ", arr));

Sort(arr);

Console.WriteLine("After sorting:");

Console.WriteLine(string.Join(", ", arr));

}

public static void Sort(int[] arr) {

int n = arr.Length;

for (int i = 0; i < n - 1; i++) {

for (int j = 0; j < n - i - 1; j++) {

if (arr[j] > arr[j + 1]) {

int temp = arr[j];

arr[j] = arr[j + 1];

arr[j + 1] = temp;

}

}

}

}

}

'''

byte\_code = '''

using System;

using System.Text;

public class Program {

public static void Main() {

byte[] bytes = {72, 101, 108, 108, 111, 32, 87, 111, 114, 108, 100};

string result = ConvertBytesToString(bytes);

Console.WriteLine(result);

}

public static string ConvertBytesToString(byte[] bytes) {

return Encoding.ASCII.GetString(bytes);

}

}

'''

byte\_code\_user='''

using System;

using System.Text;

public class Program {

public static void Main() {

Console.WriteLine("Введите строку: ");

string inputString = Console.ReadLine();

byte[] bytes = Encoding.ASCII.GetBytes(inputString);

Console.WriteLine("Массив байт:");

Console.WriteLine(string.Join(", ", bytes));

}

public static string ConvertBytesToString(byte[] bytes) {

return Encoding.ASCII.GetString(bytes);

}

}

'''

bnr\_srch = '''

using System;

public class Program {

public static void Main() {

int[] arr = {1, 2, 3, 4, 5, 6};

int searchValue = 4;

Console.WriteLine("Array: {0}", string.Join(", ", arr));

Console.WriteLine("Searching for: {0}", searchValue);

int result = BinarySearch(arr, searchValue);

if (result == -1) {

Console.WriteLine("Value not found");

} else {

Console.WriteLine("Value found at index: {0}", result);

}

}

public static int BinarySearch(int[] arr, int value) {

int left = 0;

int right = arr.Length - 1;

while (left <= right) {

int mid = (left + right) / 2;

if (arr[mid] == value) {

return mid;

} else if (arr[mid] < value) {

left = mid + 1;

} else {

right = mid - 1;

}

}

return -1;

}

}

'''

def check\_undeclared\_variables(code):

# регулярное выражение для поиска необъявленных переменных

pattern = r'\b([a-zA-Z\_]\w\*)\b'

variables = set(re.findall(pattern, code)) # получение списка всех переменных в коде

declared\_variables = {'int', 'float', 'double', 'char', 'string', 'bool'} # список зарезервированных переменных

undeclared\_variables = variables - declared\_variables # нахождение необъявленных переменных

if len(undeclared\_variables) > 0:

raise ValueError('Найдены необъявленные переменные: {}'.format(', '.join(undeclared\_variables)))

def check\_variable\_types(code):

# регулярное выражение для поиска типов переменных

pattern = r'\b(int|float|double|char|string|bool)\b\s+([a-zA-Z\_]\w\*)\s\*=?\s\*([a-zA-Z\_]\w\*)?'

variable\_matches = re.findall(pattern, code)

for match in variable\_matches:

variable\_type, variable\_name, variable\_value = match

# проверка типа переменной

if variable\_value and variable\_type not in variable\_value:

raise TypeError('Переменная {} должна иметь тип {}, а получен тип {}'.format(variable\_name, variable\_type, type(variable\_value).\_\_name\_\_))

def check\_unused\_variables(code):

# регулярное выражение для поиска используемых переменных

pattern = r'\b([a-zA-Z\_]\w\*)\b'

used\_variables = set(re.findall(pattern, code))

# регулярное выражение для поиска объявленных переменных

pattern = r'\b(int|float|double|char|string|bool)\b\s+([a-zA-Z\_]\w\*)\s\*=?'

declared\_variables = set([match[1] for match in re.findall(pattern, code)])

unused\_variables = declared\_variables - used\_variables

if len(unused\_variables) > 0:

raise ValueError('Обнаружены неиспользуемые переменные: {}'.format(', '.join(unused\_variables)))

import operator

# Доступные операторы и соответствующие им функции

OPS = {

"+": operator.add,

"-": operator.sub,

"\*": operator.mul,

"/": operator.truediv,

}

def evaluate\_expression(expr):

# Разбиваем строку на список токенов

tokens = expr.split()

# Стек для хранения операндов

stack = []

for token in tokens:

if token.isdigit():

# Если токен является числом, добавляем его на стек

stack.append(int(token))

elif token in OPS:

# Если токен является оператором, извлекаем два операнда из стека,

# применяем к ним оператор и добавляем результат на стек

op2, op1 = stack.pop(), stack.pop()

result = OPS[token](op1, op2)

stack.append(result)

else:

# Если токен не является ни числом, ни оператором, возбуждаем исключение

raise ValueError(f"Неизвестный токен: {token}")

# После обработки всех токенов результат находится на вершине стека

if len(stack) != 1:

raise ValueError("Неправильный формат выражения")

return stack.pop()

from anytree import NodeMixin, RenderTree

class AstNode(NodeMixin):

def \_\_init\_\_(self, name, parent=None, children=None):

super(AstNode, self).\_\_init\_\_()

self.name = name

self.parent = parent

if children:

self.children = children

def \_\_str\_\_(self):

return self.name

def ast\_to\_tree(ast):

if isinstance(ast, tuple):

node = AstNode(ast[0])

for sub\_ast in ast[1:]:

child = ast\_to\_tree(sub\_ast)

child.parent = node

return node

elif isinstance(ast, list):

node = AstNode("list")

for sub\_ast in ast:

child = ast\_to\_tree(sub\_ast)

child.parent = node

return node

else:

return AstNode(str(ast))

def check\_unimplemented\_methods(code):

# регулярное выражение для поиска объявления интерфейсов

pattern = r'\binterface\s+(\w+)\s\*{([^}]\*)}'

interface\_matches = re.findall(pattern, code)

for interface\_match in interface\_matches:

interface\_name, interface\_body = interface\_match

# компиляция и выполнение кода на C#

def run\_csharp\_code(code: str) -> None:

# добавление ссылки на mscorlib

clr.AddReference('mscorlib')

# компиляция исходного кода на C#

provider = System.CodeDom.Compiler.CodeDomProvider.CreateProvider('CSharp')

parameters = System.CodeDom.Compiler.CompilerParameters(['mscorlib.dll'])

result = provider.CompileAssemblyFromSource(parameters, code)

# проверка на наличие ошибок при компиляции

if result.Errors.HasErrors:

raise ValueError('Ошибка компиляции кода на C#!')

# получение экземпляра метода Main()

main\_method = result.CompiledAssembly.GetType('Program').GetMethod('Main')

# вызов метода Main() и получение результата его выполнения

main\_method.Invoke(None, None)

# запуск кода на C#

run\_csharp\_code(bnr\_srch)